UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM COMPUTAÇÃO

ADSON DE MACÊDO NASCIMENTO

GERANDO TEMPLATES PARA TESTAR MODELOS DE PLN DE FORMA AUTOMÁTICA: UM ESTUDO DE CASO COM O IMDB

CAMPINA GRANDE - PB
2022

ADSON DE MACÊDO NASCIMENTO

GERANDO TEMPLATES PARA TESTAR MODELOS DE PLN DE FORMA AUTOMÁTICA: UM ESTUDO DE CASO COM O IMDB

Trabalho de Conclusão de Curso apresentado ao Departamento do Curso Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Ciência de Computação.

Orientador: Dra. Sabrina de Figueirêdo Souto

CAMPINA GRANDE - PB
2022

N244g Nascimento, Adson de Macedo.

Gerando templates para testar modelos de PLN de forma automática [manuscrito] : um estudo de caso com o IMDB / Adson de Macedo Nascimento. - 2022.

45 p. : il. colorido.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2022.

"Orientação : Profa. Dra. Sabrina de Figueirêdo Souto , Coordenação do Curso de Computação - CCT."

1. Aprendizagem de máquina. 2. Processamento de Linguagem Natural. 3. Ferramenta de testes. 4. Técnica de aprendizagem. I. Título

21. ed. CDD 600

Elaborada por Deivid A. M. Neves - CRB - 15/899 BC/UEPB

ADSON DE MACÊDO NASCIMENTO

GERANDO TEMPLATES PARA TESTAR MODELOS DE PLN DE FORMA AUTOMÁTICA: UM ESTUDO DE CASO COM O IMDB

Trabalho de Conclusão de Curso apresentado ao Departamento do Curso Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Ciência de Computação.

Trabalho aprovado em 23 de Março de 2022.

BANCA EXAMINADORA

Sabrina de F. Souto.

Profa. Dra. Sabrina Figueiredo Souto (DC - UEPB)
Orientador(a)

Melina Mongiovi

Profa. Dra. Melina Mongiovi (UFCG)
Examinador(a)

Kézia de Vasconcelos Oliveira Santos

Profa. Dra. Kézia Vasconcelos (DC - UEPB)

A toda minha família, que sempre me apoiou durante esta minha jornada, em especial à minha esposa Leidiana Rodrigues, e filhos, Douglas Rodrigues e Júlia Rodrigues, DEDICO.

RESUMO

O uso de sistemas que utilizam alguma técnica de aprendizagem de máquina tem crescido bastante nos últimos anos, o que levou ao desenvolvimento de novas técnicas e aprimoramento de técnicas existentes. Dentro da aprendizagem de máquina encontramos diversas sub-áreas, e mesmo dentro das sub-áreas existem diversas tarefas resolvidas com diferentes técnicas. Uma das sub-áreas mais pesquisadas recentemente é a área de Processamento de Linguagem Natural, ou PLN. As técnicas de PLN são baseadas em compreensão textual por parte dos sistemas, geralmente levando-se em conta o contexto. Devido ao fato de, muitas vezes, esses sistemas lidarem com problemas críticos, surgiram preocupações quanto a qualidade do software desenvolvido. No entanto, os processos da engenharia de software tradicional não se aplicam diretamente a esse tipo de software. Além disso, a grande diversidade e especificidade das técnicas dificultam a criação de ferramentas de teste que se aplicam de forma geral. Sendo assim, existem diversas ferramentas que auxiliam no processo de testes em sistemas de aprendizagem de máquina, muitas delas lidando com tipos específicos de sistemas. No entanto, algumas delas ainda dependem de interação humana, requerendo criatividade por parte do usuário para que sua utilização se torne viável. Neste trabalho propomos uma abordagem para automatizar algumas dessas etapas manuais, através de um algoritmo que gera templates que servem como base para criação de casos de teste para ferramentas de testes. Através de um experimento prático, conseguimos extrair templates a partir de um dataset de testes do IMDB e os utilizamos juntamente com o Checklist para encontrar falhas em um modelo BERT de análise de sentimentos.

Palavras-chaves: Aprendizagem de Máquina, Processamento de Linguagem Natural, Ferramenta de testes, Técnica de aprendizagem.

ABSTRACT

The use of systems based in some machine learning technique has grown a lot in recent years, which has led to the developing new techniques and improving existing techniques. Within machine learning area we find several sub-areas, and even within the sub-areas there are several tasks solved with different techniques. One of the most researched sub-areas recently is the Natural Language Processing, or NLP. The NLP techniques are based on textual understanding by the systems, generally considering the context. Due to the fact that these systems often deal with critical problems, concerns have arisen about developed software quality. However, traditional software engineering processes do not directly apply to this type of software. In addition, the great diversity and specificity of techniques make it difficult to create generally applicable testing tools. Therefore, there are several tools that help in the testing process in machine learning systems, many of them dealing with specific system kind. However, some of them still depend on human interaction, requiring user creativity so that their use becomes viable. In this work we propose an approach to automate some of these manual steps, through an algorithm that generates templates that serve as a basis to creating test cases for testing tools. Through a practical experiment, we extracted templates from an IMDB test dataset and used them together with Checklist to find flaws in a BERT sentiment analysis model.

Keywords: Machine Learning, Natural Language Processing, Testing tool, Learning technique.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LIME	Local Interpretable Model-Agnostic Explanations
ML	Machine Learning
NER	Named Entity Recognition
PLN	Processamento de linguagem natural

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, diversos sistemas que utilizam algum tipo de aprendizagem de máquina (ou *Machine Learning*) têm sido desenvolvidos. Cenários que num passado recente eram pura ficção científica começam a figurar no cotidiano das pessoas. Robôs que conversam com pessoas, carros que se movem de forma autônoma, sistemas de monitoramento que identificam pessoas, etc. Esses sistemas são capazes de realizar tais tarefas com uma certa precisão de acerto, normalmente muito próximo da exatidão.

Uma das subáreas da aprendizagem de máquina é o Processamento de Linguagem Natural, ou PLN (SAS, 2022). Aplicações como tradutores de texto modernos, buscadores de conteúdo, transcrição da linguagem falada-escrita, entre muitos outros, são todos baseados em modelos de PLN.

Diversas técnicas têm sido desenvolvidas para resolver tarefas de PLN. Uma das mais bem sucedidas e utilizadas no mundo é modelo BERT (DEVLIN et al., 2019) desenvolvido por pesquisadores do Google. Este modelo possui uma característica importante de permitir o uso para várias tarefas diferentes.

A crescente adoção por este tipo de sistema tem contribuído consideravelmente para a evolução de diversas áreas de estudo e conhecimento. Aplicações na área da medicina, por exemplo, têm proporcionado melhor desempenho nos processos de diagnóstico, prognóstico e tratamento, oferecendo menos riscos ao paciente (BRAGA et al., 2019). Muitos desses sistemas são considerados críticos por lidar com dados sensíveis ou até mesmo por oferecerem risco à vida em caso de falhas.

Dentro da Engenharia de Software tradicional, existem processos dedicados a identificar e mitigar esses riscos, bem como detecção e correção de falhas (SOMMERVILLE, 2011). No contexto de aprendizagem de máquina, como não poderia ser diferente, também existem alguns métodos para avaliação dos modelos (RICCIO et al., 2020).

Avaliar um modelo de aprendizagem de máquina não é uma tarefa das mais simples. Uma das principais razões é a forma diferente de como esses sistemas são construídos quando comparados com sistemas tradicionais (ZHANG et al., 2020). Outra grande dificuldade (ou consequência da anterior) é a ausência de técnicas que se apliquem a vários tipos de sistemas de aprendizagem de máquina simultaneamente.

Tradicionalmente, são usadas algumas métricas para analisar os resultados, avaliar o desempenho e validar os modelos, tais como acurácia, matriz de confusão, recall, precisão e f1-score (MINAEE, 2019). Essas métricas nos dão bastante informação sobre o modelo, sendo, normalmente, suficiente para uma primeira análise da performance do modelo.

Algumas vezes, os modelos podem ser bem avaliados ao usarmos uma determinada métrica, mas se saírem muito mal quando utilizamos uma outra métrica. Isso pode acontecer por diversas razões. Assim, uma avaliação com diferentes métricas pode nos dar uma informação mais precisa sobre a qualidade do modelo. Além disso, características não desejáveis, mas que

podem estar presentes nos modelos, como *overfitting* (YING, 2019) (quando o modelo está muito ajustado aos dados de treinamento) por exemplo, podem acabar mascarando defeitos existentes no modelo.

Algumas ferramentas desenvolvidas para avaliação de modelos utilizam diferentes estratégias para avaliar o modelo. No contexto de **PLN** podemos destacar o **LIME** (RIBEIRO; SINGH; GUESTRIN, 2016) - que utiliza explanações para avaliar as predições do modelo, **TextAttack** (MORRIS et al., 2020) - realiza ataques adversariais em *datasets* juntamente como o modelo, e **CheckList** (RIBEIRO et al., 2020) - cria suítes de teste com foco em capacidades linguísticas.

Como vimos, o testes para sistemas **PLN** são um pouco mais complexos que os testes de software tradicionais. Contudo, existem algumas ferramentas já desenvolvidas com o propósito de auxiliar nessa tarefa.

Embora a proposta da maioria das ferramentas de teste seja automatizar ao máximo o processo de testes, algumas delas possuem alguns passos que ainda exigem interação humana. Um exemplo disso é a criação de *templates* para geração de dados de teste. Basicamente, um *template* é uma "receita" para geração de dados de teste (RIBEIRO et al., 2020). Este conceito será explicado com mais detalhes em seções futuras.

A criação manual de *templates*, além de poder levar a problemas como viés por exemplo, pode ser uma tarefa um tanto trabalhosa. Frequentemente, o design de *templates* requer anotadores com expertise na linguagem compreendida pelo modelo, como no trabalho realizado por (BHATT et al., 2021).

Outra parte deste mesmo problema está no próprio uso do *template*. Embora os templates contribuam com o aumento na geração de casos de teste, o seu uso ainda depende da criatividade dos usuários para escolher as palavras a serem substituídas em cada espaço reservado (RIBEIRO et al., 2020).

A proposta de solução para o problema é basicamente automatizar o processo de criação de *templates*. Para isso, propomos desenvolver um algoritmo que consiga extrair *templates* a partir de dados provenientes dos próprios *datasets* de testes do modelo. A solução deve também resolver a questão do uso do *template*, automatizando também a geração de palavras para substituição. A ideia é usar como entrada um conjunto de dados textuais, dados esses que também sirvam como entrada para o modelo alvo, e, através de transformações e filtros, encontrar as melhores sentenças que possam servir de *templates* para gerar casos de teste para o modelo.

1.1 Objetivos

Este trabalho tem como objetivo principal propor uma abordagem para servir de auxílio para ferramentas de teste de sistemas PLN que utilizam templates para geração de casos teste. Este auxílio é feito através da geração de *templates* de forma automatizada a partir de dados provenientes de *datasets*.

Com base no nosso objetivo geral, elencamos os seguintes objetivos específicos.

1. Mensurar e identificar as principais características que devem ser levadas em consideração para a construção dos templates;
2. Desenvolver uma abordagem para o algoritmo que seja capaz de extrair os *templates* a partir de dados provenientes de um *dataset* de entrada;
3. Aplicar nossa abordagem em um estudo de caso com o **IMDB**.

1.2 Estrutura do documento

O capítulo 2 traz um apanhado teórico, que servirá de base de conhecimento para a compreensão do restante do trabalho. No capítulo 3, trataremos dos detalhes de como foi desenvolvida a abordagem, explorando o algoritmo e suas principais etapas. O capítulo 4 é dedicado ao estudo de caso com o **IMDB**, onde mostraremos os resultados obtidos e faremos uma breve discussão sobre eles. No capítulo 5 apresentaremos as conclusões sobre o que foi desenvolvido durante o trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo serão discutidos temas necessários para compreensão do trabalho. Inicialmente falaremos sobre *Machine Learning* e suas aplicações. Em seguida, veremos conceitos sobre Processamento de Linguagem Natural (PLN). Após isso, discutiremos um pouco sobre testes para modelos de *Machine Learning*, bem como os conceitos de abordagens *black-box* e *white-box*. Falaremos também de técnicas de teste específicas para PLN e, por fim, abordaremos algumas ferramentas de avaliação existentes na literatura.

2.1 Machine Learning

Aprendizagem de máquina (ou *Machine Learning*) é uma subárea da Inteligência Artificial (IA), e refere-se a um conjunto de técnicas de modelagem estatística que alimentam o entusiasmo recente no mercado de softwares e serviços (AMERSHI et al., 2019). Cada vez mais as aplicações que dispõem de algum recurso de aprendizagem de máquina estão presentes no cotidiano das pessoas, desde mecanismos de busca até aplicações envolvendo visão computacional.

A crescente adoção por este tipo de sistema tem contribuído consideravelmente para a evolução de diversas áreas de conhecimento. Aplicações na área da medicina, por exemplo, têm proporcionado melhor desempenho nos processos de diagnóstico, prognóstico e tratamento, oferecendo menos riscos ao paciente (BRAGA et al., 2019).

Assim como no desenvolvimento dos softwares tradicionais, que faz o uso de processos definidos pela engenharia de software, existem também processos que definem um fluxo de trabalho para desenvolvimento de modelos de aprendizagem de máquina. Dentre os estágios deste fluxo podemos identificar algumas fases importantes: definição dos requisitos do modelo, coleta de dados, tratamento de dados, definição do modelo, treinamento, avaliação e monitoramento do modelo.

Existem na literatura diversas tarefas, podendo ser preditivas (onde o modelo realiza uma previsão para uma determinada entrada) ou descritivas (quando o modelo realiza uma descrição da entrada, como uma segmentação, por exemplo), que são resolvidas usando técnicas de *Machine Learning*. Essas técnicas de podem ser divididas em dois grandes grupos:

> **Aprendizado supervisionado:** O aprendizado supervisionado é uma abordagem de aprendizado de máquina definida pelo uso de conjuntos de dados rotulados (DELUA, 2021), isto é, o conjunto de dados de treinamento inclui as respostas corretas para cada entrada. Esses conjuntos de dados são projetados para treinar ou "supervisionar" algoritmos para classificar dados ou prever resultados com precisão. Usando entradas e saídas rotuladas, o modelo pode medir sua precisão e aprender com o tempo (DELUA, 2021). Os problemas resolvidos com este tipo de abordagem são de classificação (o modelo atribui uma categoria para cada entrada dada) e de regressão (o modelo realiza uma previsão de um valor

baseada na relação entre as variáveis dependentes).

Aprendizado não-supervisionado: Diferentemente do aprendizado supervisionado, o aprendizado não supervisionado usa algoritmos de aprendizado de máquina para analisar e agrupar conjuntos de dados não rotulados (DELUA, 2021). Os algoritmos buscam padrões e realizam agrupamentos nos dados. Uma das tarefas resolvidas com essa abordagem é a clusterização, que consiste em agrupar os dados de acordo com propriedades semelhantes.

2.2 Processamento de Linguagem Natural

O Processamento de Linguagem Natural (PLN) é uma importante subárea de Machine Learning e tem sido amplamente utilizado em aplicações do cotidiano. Como parte fundamental deste trabalho, é necessário conhecer alguns conceitos sobre PLN.

Primeiro, é preciso compreender que, de maneira geral, PLN é um conjunto de técnicas que ajudam o computador a lidar com a linguagem humana, permitindo sua manipulação e entendimento. Tais técnicas são extremamente importantes, já que as máquinas não são capazes de compreender diretamente nossa linguagem.

O desenvolvimento e aprimoramento de PLN possibilitou que hoje tivéssemos sistemas inteligentes que compreendem a linguagem humana com excelência, o que tornou seu uso altamente difundido. Por exemplo, quando fazemos perguntas a um smartfone através da linguagem falada e esta é compreendida e transformada em uma linguagem escrita, ou quando buscadores inteligentes sugerem complementos e exibem resultados mais precisos, técnicas de PLN estão sendo utilizadas.

Recentemente, técnicas baseadas em aprendizado não-supervisionado e modelos pré treinados têm sido responsáveis por grandes avanços na área de PLN (OPENAI, 2018). Uma das inovações mais recentes é modelo **BERT (Bidirectional Encoder Representations for Transformers)**, desenvolvido por pesquisadores do Google (DEVLIN et al., 2019). Este modelo tem por característica o uso de transformadores bi-direcionais, que capturam o contexto ao utilizar um modelo de linguagem mascarada.

Uma versão aprimorada do **BERT** foi proposto por(LIU et al., 2019). O modelo **RoBERTa (Robustly optimized BERT approach)** introduz aprimoramentos no treinamento do **BERT** ao avaliar cuidadosamente os efeitos dos hiper-parâmetros e quantidade de dados no treinamento do modelo (LIU et al., 2019).

Com o objetivo de diminuir a complexidade do **BERT**, (SANH et al., 2019) apresentou o modelo **DistilBERT**, uma versão "destilada"do **BERT** (menos complexa) que possui uma performance muito próxima. Experimentos mostraram que, para um conjunto de 9 tarefas, o **DistilBERT** conseguiu reter 97% da performance do **BERT** original com 40% menos parâmetros (SANH et al., 2019).

Outro modelo baseado em transformadores é o **XLNET** (YANG et al., 2019) que, em experimento de comparação, superou consistentemente o **BERT** em um amplo espectro de

problemas (YANG et al., 2019). A principal diferença estrutural entre o **BERT** e o **XLNET** é que o **BERT** é categorizado como um modelo de linguagem de *AutoEncoder*, que visa reconstruir dados de uma entrada corrompida, ao passo que o **XLNET** é um modelo de linguagem *AutoRegressive*, que utiliza o contexto para prever a próxima palavra.

2.3 Testes de sistemas de ML

Avaliar um modelo de ML não é uma tarefa das mais simples. Uma das principais razões é a "natureza"e construção diferentes dos sistemas de ML quando comparados com sistemas tradicionais (ZHANG et al., 2020). Outra grande dificuldade (ou consequência da anterior) é a ausência de técnicas que se apliquem a vários tipos de sistemas de ML simultaneamente. A divisão menos óbvia do sistema em componentes, fazendo com que os defeitos só possam ser compreendidos analisando o sistema como um todo, torna os testes ainda mais difícil (ZHANG et al., 2020).

Assim como em softwares tradicionais, em que nós definimos alguns atributos de qualidade do software, para a avaliação dos modelos isto também é necessário. Alguns importantes atributos de qualidade de modelos ML são definidos no trabalho de Zhang (ZHANG et al., 2020):

Corretude - se refere à probabilidade do sistema ML realizar a tarefa da maneira correta;

Grau de overfitting: - define o quanto o modelo está ajustado demais aos dados usados em treinamento;

Robustez - mede a resiliência do modelo diante de perturbações nos dados;

Segurança - é a resiliência do sistema contra danos causados por meio da manipulação ou acesso ilegal aos componentes de ML;

Privacidade dos dados - é a habilidade de manter os dados de forma privada;

Eficiência - se refere a velocidade de construção e predição do modelo de ML;

Justiça - qualidade que garante que o modelo não tomará decisões de forma discriminatória (em relação a raça, gênero, religião, etc);

Interpretabilidade - se refere ao grau em que um observador consegue compreender os motivos da decisão tomada pelo sistema de ML.

Embora todos esses atributos sejam importantes, a proposta deste trabalho trará um enfoque maior no atributo **robustez** do modelo, visto que este está mais relacionado com a proposta.

2.3.1 Black-box

O teste de software conhecido como **black-box**, ou caixa preta, é o tipo de teste em que o código do software não é conhecido. Em outras palavras, o foco do teste não é no código que constitui o software, mas sim no seu funcionamento.

Quando aplicado a modelos de aprendizado de máquina, o teste de caixa preta significaria testar modelos de aprendizado de máquina sem conhecer os detalhes internos, como recursos do modelo de aprendizado de máquina, o algoritmo usado para criar o modelo, etc. O desafio, no entanto, é identificar o oráculo de teste que poderia verificar o resultado do teste em relação aos valores esperados conhecidos de antemão. (KUMAR, 2018).

Alguns testes de modelos de ML que seguem a abordagem caixa preta são: **teste de performance**, que consistem em, utilizando algumas métricas (ver Seção 2.4.1), analisar os resultados do modelo quando submetido a dados de teste ou novos dados; **teste metamórfico**, onde as entradas são modificadas de forma certa forma que se conheça o comportamento esperado da saída; **codificação dupla**, que consiste em construir diferentes modelos usando diferentes algoritmos e comparar os resultados obtidos quando submetidos ao mesmo conjunto de dados de teste.

2.3.2 White-box

A abordagem de teste conhecida como **white-box**, ou caixa branca, é baseada analisar a estrutura do código do software em vez focar em funcionalidades. Em softwares tradicionais são usados como métricas a cobertura de de código, cobertura de caminhos, cobertura de sentenças, entre outras.

No contexto de aprendizagem de máquina podemos destacar alguns critérios como no trabalho de (PEI et al., 2017), que propôs o **DeepXplore**, uma abordagem de teste caixa branca que utiliza a cobertura de neurônios como métrica. Já em (KIM; FELDT; YOO, 2018), o autor propõe critérios que medem a adequação do modelo quando submetido a uma entrada considerada “surpreendente” em relação aos dados de treino.

2.4 Avaliação de sistemas de PLN

É preocupante que, embora as tecnologias de linguagem natural estejam se tornando cada vez mais difundidas em nossas vidas cotidianas, haja pouca garantia de que esses sistemas de PLN não falharão catastroficamente ou amplificarão a discriminação contra dados demográficos minoritários quando expostos a informações de fora da distribuição de treinamento (TAN et al., 2021).

Nesse contexto, os modelos têm se tornando cada vez mais frágeis ou menos úteis em situações do mundo real. Isso pode ser devido a vários fatores, como complexidade e variabilidade da linguagem, a diferença entre treinamento, teste e dados do mundo real e compreensão insuficiente das capacidades e limitações do próprio modelo (BHATT et al., 2021). Um exemplo

emblemático é o do GPT-3 (BROWN et al., 2020), que sugere que o paciente pesquise no *YouTube* para realizar seu próprio diagnóstico (ROUSSEAU; BAUDELAIRE; RIERA, 2020), além de concordar com o suicídio do paciente (ROUSSEAU; BAUDELAIRE; RIERA, 2020).

Pesquisas recentes têm evidenciado que avaliar os modelos com *datasets* de teste e usando apenas métricas, como acurácia por exemplo, pode não representar um bom indicativo da qualidade do modelo. Isso porque os dados no mundo real normalmente são significativamente diferentes dos dados usados no treinamento e avaliação (RIBEIRO; SINGH; GUESTRIN, 2016). Assim, algumas técnicas adicionais para testar modelos de PLN são necessárias. Cada uma dessas técnicas pode endereçar um ou mais atributos mencionados na seção 2.3. Nas seções seguintes abordaremos algumas dessas técnicas.

2.4.1 Métricas de avaliação

Durante a fase de treinamento dos modelos, eles são frequentemente submetidos a avaliações usando algumas métricas. Essas métricas são normalmente usadas para decidir sobre a qualidade de um modelo em relação a outro. As principais métricas utilizadas são:

Acurácia: Esta é uma das primeiras métricas usadas e indica o percentual de acertos em relação ao número de entradas de teste.

Matriz de confusão: Esta é métrica para classificadores. Com ela é possível ter uma visualização de forma matricial onde podemos identificar, para cada possível classe, quantas entradas foram preditas de forma correta e quantas foram preditas como cada uma das outras opções.

Recall: O *recall* é uma métrica que informa o quanto o classificador consegue acertar para uma determinada classe. Esta métrica é expressa pela fórmula:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Precisão: A precisão de um classificador define o quanto do que foi predito para uma determinada classe está correto. Ela pode ser expressa pela fórmula:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

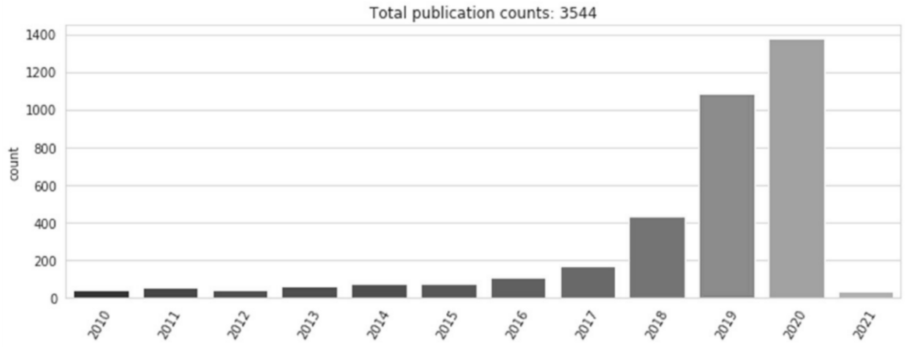
F1-score: O F1-score é uma métrica que representa um equilíbrio entre o *recall* e a precisão. Ela é definida pela fórmula:

$$F1_{score} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

2.4.2 Explanações

Uma das grandes diferenças entre os sistemas de ML em relação aos softwares tradicionais é sua natureza *black-box*, e isso dificulta a interpretabilidade do modelo. Os modelos, embora sejam muito bons em realizar predições, geralmente não fornecem explicações para suas previsões de maneira que humanos possam entender facilmente (ZHOU, 2021).

Figura 1 – Número de publicações de pesquisa sobre explanações de ML (com base em Scopus.com até Dezembro de 2020)



Fonte: Bhatt et al. (2021).

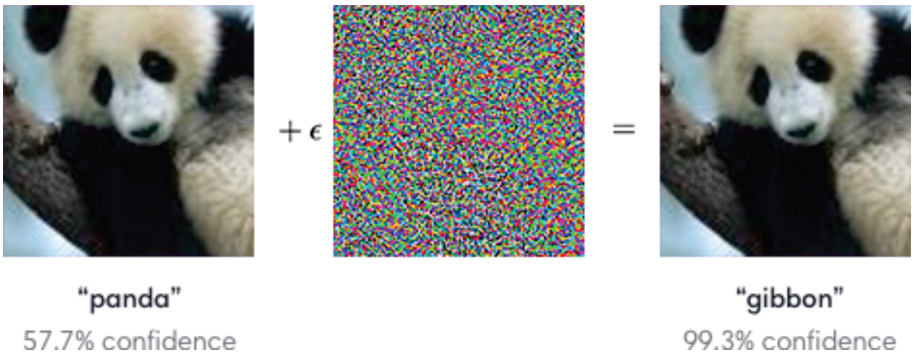
Desta maneira, as técnicas de **explanação** tentam resolver este problema, permitindo que um usuário possa compreender melhor as razões pelas quais o modelo tomou uma determinada decisão. Explanar uma previsão significa apresentar artefatos textuais ou visuais que fornecem compreensão qualitativa da relação entre os componentes da instância (por exemplo, palavras em texto, manchas em uma imagem) e a previsão do modelo (RIBEIRO; SINGH; GUESTRIN, 2016).

Recentemente, o número de pesquisas com foco em técnicas de explanações tem crescido, como mostra a Figura 1. No entanto, a maioria das pesquisas no campo de explanações focam em criar novas técnicas para melhorar a explanabilidade dos modelos (BHATT et al., 2021).

2.4.3 Ataque adversarial

Para compreender o que é um **ataque adversarial** precisamos definir o que é um **exemplo adversarial**. Exemplos adversariais são entradas para modelos de aprendizado de máquina projetadas intencionalmente para fazer com que o modelo cometa um erro; são como ilusões de ótica para máquinas (OPENAI, 2017). Normalmente, esses exemplos adversariais são produzidos a partir de pequenas perturbações em entradas existentes. O **ataque adversarial** é, portanto, a técnica de geração de exemplos adversariais.

Figura 2 – Exemplo adversarial



Fonte: (OPENAI, 2017).

A Figura 2 ilustra, para uma suposta tarefa de reconhecimento de imagens, um exemplo adversarial gerado a partir da adição de uma pequena quantidade de ruído a uma imagem original.

Para o contexto de PLN existem diversas técnicas de ataque adversarial na literatura. (GAO et al., 2018a) propôs o **DeepWordBug** para geração de exemplos adversariais ao simular erros de digitação no texto. Já (GAO et al., 2018b) propôs gerar exemplos adversariais ao remover palavras não importantes da entrada. Em (GARG; RAMAKRISHNAN, 2020), o autor propõe **BAE (BERT-based Adversarial Examples)**, que gera exemplos adversariais ao adicionar ou substituir *tokens* do texto usando um modelo de linguagem mascarada.

2.4.4 Aumento de dados

O crescimento do uso de PLN e a exploração de novas tarefas, muitas vezes com poucos exemplos para o treinamento do modelo criam um cenário onde o uso de técnicas de **aumento de dados** exerce um papel extremamente importante. **Aumento de dados** refere-se a estratégias para aumentar a diversidade de exemplos de treinamento sem coletar explicitamente novos dados (FENG et al., 2021).

Essas estratégias normalmente baseiam-se em modificar entradas existentes de modo a conservar a semântica e adicioná-las ao conjunto de dados de treinamento. O aumento de dados podem ser importante também para corrigir problemas de *overfitting*, causado pela escassez de exemplos para o treinamento.

2.4.5 Treino adversarial

O procedimento de **treino adversarial** pode ser visto como minimizar o erro de pior caso quando os dados são perturbados adversamente (GOODFELLOW; SHLENS; SZEGEDY, 2014). Através do treino adversarial, podemos alcançar uma maior robustez e uma melhor generalização do modelo.

Normalmente, uma técnica de treino adversarial inclui etapas como treinamento inicial com os dados originais (limpos), em seguida os dados são substituídos por uma versão com perturbações e o modelo retreinado. Esta versão perturbada do conjunto de dados é substituída

pelo original e é regenerada periodicamente de acordo com as fraquezas atuais do modelo (MORRIS et al., 2020).

2.5 Ferramentas para testes em sistemas de Machine Learning

Algumas ferramentas de teste para ML existentes utilizam diferentes estratégias para alcançar essas qualidades. Cada ferramenta pode focar em uma ou mais propriedades do modelo. Nas seções seguintes, abordaremos algumas das principais ferramentas de teste para ML.

2.5.1 LIME

O principal objetivo do LIME (RIBEIRO; SINGH; GUESTRIN, 2016) é proporcionar uma melhor interpretabilidade das predições realizadas pelos modelos. A estratégia usada pela ferramenta é explicar, através de explanações (ver seção 2.4.2), quais foram as *features* mais relevantes na predição de uma instância quando submetida a um classificador.

Com as explanações das predições podemos ter um melhor entendimento e, consequentemente, melhor confiabilidade no classificador. Neste contexto, um profissional estaria mais seguro em tomar uma decisão baseada na predição do classificador mediante uma explicação de "como" o classificador chegou a tal predição. Abaixo, algumas características que tornam o LIME uma ferramenta poderosa para avaliação de modelos de *Machine Learning*:

Interpretável: permitindo que uma pessoa, mesmo que não seja expert em *Machine Learning*, consiga entender o "porquê" de uma predição;

Fidelidade local: LIME tenta replicar o comportamento do classificador nas proximidades da instância que está sendo prevista;

Agnóstico ao modelo: a ferramenta é capaz fornecer explanações para qualquer classificador, já que constrói internamente um modelo que se comporta (localmente) como o modelo explanado;

Perspectiva global: com um conjunto representativo de explanações o usuário consegue ter uma intuição global sobre o modelo. Para alcançar este objetivo, LIME dispõe de um algoritmo para selecionar de forma automática o conjunto de instâncias a serem explanadas.

2.5.2 TextAttack

O TextAttack (MORRIS et al., 2020) é um *framework* desenvolvido em Python para gerar ataques adversariais, treino adversarial e aumento de dados em sistemas de *Machine Learning* com processamento de linguagem natural.

Um **ataque adversarial** consiste em, dados um modelo classificador e um conjunto de dados de entrada, realizar pequenas perturbações nesses dados de modo a levar a uma classificação incorreta pelo modelo.

Com a ferramenta é possível usar "receitas" de ataque ou montar configurações de ataque personalizadas. Isto é possível graças à estrutura interna do *framework*, que divide a tarefa em 4 sub-componentes:

Função objetiva: objetivo do ataque, como gerar uma classificação incorreta por exemplo;

Método de busca: algoritmo usado dada uma instância de entrada, buscar uma nova instância transformada baseando-se na função objetiva escolhida;

Transformações: função que transforma a instância de entrada, causando pequenas perturbações (como remover ou adicionar um caractere por exemplo);

Restrições: funções que determinam se a instância modificada ainda é válida (restrições sintáticas por exemplo)

O *framework* já dispõe de várias implementações (providos da literatura) para cada um dos sub-componentes, permitindo que façamos diferentes combinações entre eles ("receitas" personalizadas).

As principais aplicações da ferramenta incluem: realização de **ataque adversarial** - a ferramenta gera, a partir de um conjunto de entradas, um outro conjunto de instâncias que geram classificações incorretas; o **treino adversarial** - a ferramenta realiza o treinamento do modelo usando, após um treino inicial com os exemplos originais, exemplos adversariais gerados a partir dos dados originais; aumento de dados - a ferramenta gera exemplos através da adição de perturbações em dados existentes para serem adicionados aos dados de treino já existentes.

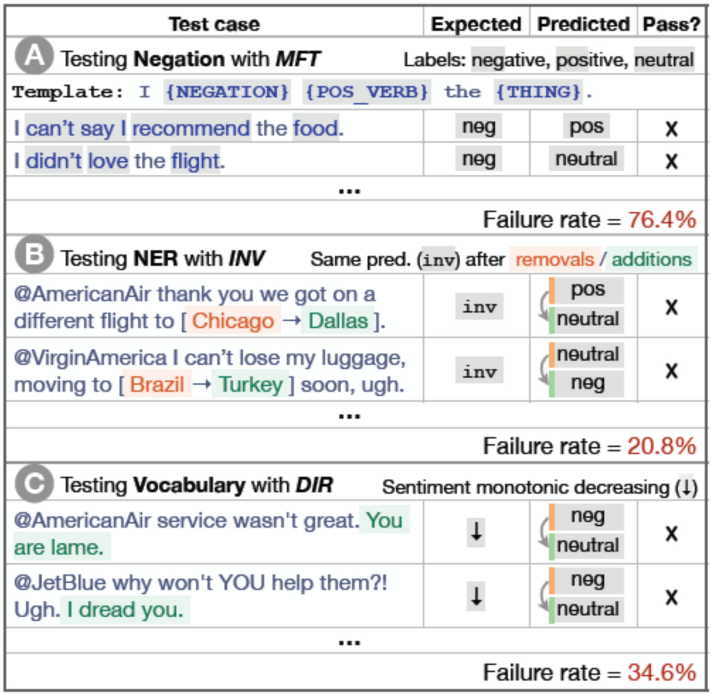
O TextAttack também permite criar implementações personalizadas dos sub-componentes, o que o torna uma excelente ferramenta para auxiliar pesquisadores a desenvolver protótipos para novas estratégias de ataques adversariais.

2.5.3 CheckList

O CheckList (RIBEIRO et al., 2020) é um *framework* para desenvolvimento de testes e geração de dados de teste para sistemas de *Machine Learning* com processamento de linguagem natural. A metodologia usada pelo CheckList é inspirada em princípios de **testes comportamentais** da engenharia de software.

Os testes desenvolvidos usando o CheckList são baseados em capacidades linguísticas dos modelos. Com isso, podemos gerar suítes de teste com vários cenários de teste de forma sistemática, avaliando as diferentes capacidades linguísticas do modelo como: robustez (capacidade de lidar com erros de digitação ou acréscimo de informação irrelevante, por exemplo), negação, reconhecimento de entidade nomeada, tempo verbal, correferência, etc.

Figura 3 – Utilizando o CheckList em um modelo de análise de sentimento



Fonte: (RIBEIRO et al., 2020).

A ferramenta permite que, para cada capacidade linguística do modelo, possamos realizar vários testes de três possíveis tipos:

> **MFT (Minimum Functionality test):** inspirado em testes de unidade da engenharia de software, testa cada capacidade linguística de forma simples e direta;
>
> **INV (Invariance):** verifica se pequenas alterações nos dados de teste produzem variações nas predições;
>
> **DIR (Directional Expectation test):** verifica o comportamento da predição ao inserirmos pequenas perturbações;

Na Figura 3 podemos ver um exemplo de utilização da ferramenta. O caso de teste **A** representa um teste **MFT** para capacidade linguística **negação**. O teste **B** é do tipo **INV** para capacidade linguística **NER**. E o caso de teste **C** é um teste do tipo **DIR** para capacidade linguística **vocabulário**.

Além de permitir realizar transformações em dados existentes, o CheckList permite geração de dados de forma rápida através de *templates* e *lexicons*. Por exemplo: com um *template* do tipo "The movie {MOVIE} was {POS_ADJ}"e os conjuntos de *lexicons* MOVIE=["Titanic", "MIB"] e POS_ADJ=["amazing", "excelent", "good"], gerariam 6 exemplos, formados pelas combinações de cada uma das opções dos *lexicons* substituídos no *template*.

Como podemos ver, a versatilidade ao gerar casos de teste para diversas capacidades do modelo é o que torna esta ferramenta tão interessante. No entanto, a geração de dados por

templates pode ser uma tarefa um tanto trabalhosa. Frequentemente, o design de *templates* requer anotadores com expertise na linguagem compreendida pelo modelo, como no trabalho realizado por (BHATT et al., 2021).

3 ABORDAGEM

Como vimos nas seções anteriores, exitem algumas ferramentas de teste para modelos de ML com diferentes abordagens. Em particular, a ferramenta CheckList (RIBEIRO et al., 2020) é capaz de realizar testes baseados em capacidades linguísticas do modelo (ver Seção 2.5.3). Alguns desses testes são implementados com o auxílio de *templates*, que são criados manualmente por anotadores e usados para geração de dados de teste.

A elaboração de *templates* para geração de dados de teste do zero por anotadores humanos, no entanto, pode ser uma tarefa bastante difícil, visto que frequentemente requer expertise na linguagem com a qual o modelo vai lidar, além de estar sujeito a criatividade do anotador (BHATT et al., 2021).

Neste capítulo falaremos de como poderíamos gerar *templates* de forma automatizada. Iniciaremos por uma visão geral da estratégia. Nas seções seguintes trataremos das diferentes abordagens propostas para geração dos *templates*.

3.1 Visão geral

O principal objetivo deste trabalho é gerar casos de testes a partir de *templates* gerados de forma automatizada. A ideia, na verdade, baseia-se em gerar os templates a partir de um conjunto de dados de teste existente, extraindo templates com foco em capacidades linguísticas do modelo.

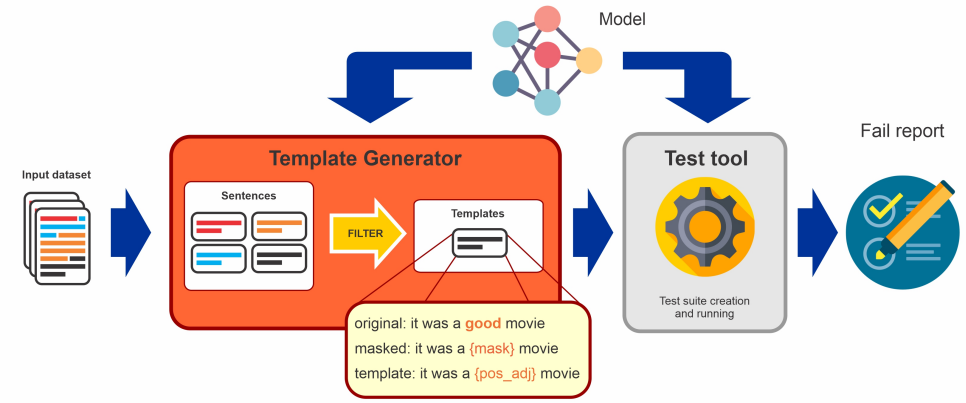
A ideia de *Capacidade Linguística do Modelo*, introduzida na metodologia de testes do **CheckList** (RIBEIRO et al., 2020), é basicamente a capacidade de um modelo PLN lidar com determinadas características da linguagem. Por exemplo, considere um cenário onde nós temos um modelo PLN que classifica textos em *positivos* ou *negativos* (análise de sentimentos). Uma das capacidades linguísticas que este modelo deve possuir é a capacidade de lidar com a negação de frases *positivas* ou *negativas* (capacidade **Negação**). Outra capacidade linguística poderia ser a capacidade de lidar com a troca de palavras por sinônimos (capacidade **Vocabulário**).

Nesse contexto, os templates são usados para servir como "moldes" para frases (ou até mesmo textos maiores), onde algumas palavras especiais dentro do template (*lexicons*) são, durante o processo de testes, substituídas por outras para gerar os casos de teste. Os *lexicons* são identificados dentro do template por palavras entre chaves, como por exemplo {pos_adj}, que indicam adjetivos positivos.

Para realizar a tarefa de geração de *templates* de forma automatizada, foi desenvolvido o algoritmo **TemplateGenerator**, que posteriormente foi utilizado como base para cinco diferentes variações.

De modo geral, podemos interpretar o **TemplateGenerator** como sendo uma função $G(x)$ que recebe como entrada um conjunto de instâncias x e, após submetê-las a uma série de filtros e transformações, gera como saída um conjunto de templates $\mathbb{T}$ e um conjunto de lexicons $\mathbb{L}$. Na Figura 4 podemos ver uma abstração do uso do **TemplateGenerator**.

Figura 4 – Gerando templates com o Template Generator



Fonte: Autoria própria.

A figura mostra os *templates* sendo extraídos pelo **TemplateGenerator** (representando pelo retângulo laranja) a partir de um *dataset* de entrada e o modelo PLN alvo. Em seguida, os templates, juntamente com os lexicons gerados, são utilizados como entrada para uma ferramenta de testes (representada pelo retângulo em cinza), usada para geração e execução de testes do modelo PLN alvo, produzindo o relatório de falhas.

A entrada para o algoritmo é um conjunto de instâncias em formato de texto previamente rotuladas. Um exemplo de entrada para o algoritmo pode ser vista na Figura 5, onde a coluna *label* indica o rótulo da instância. Neste exemplo, a instância é uma entrada para um modelo de análise de sentimentos e está rotulada como classe 1, ou seja, *positiva*.

Figura 5 – Exemplo de instância de entrada

Text	Label
This is a really good flick with awesome humor. I bought this movie right after i saw it. Great directing, good script, worth renting.	1 (pos)

Fonte: Autoria própria.

Os passos do algoritmo estão descritos no Algoritmo 1. Este algoritmo possui três etapas bem definidas: a divisão das instâncias em sentenças (linhas 4-11 do algoritmo), etapa de filtragem (linha 12 do algoritmo) e etapa de geração dos templates (linhas 13-18 do algoritmo). Cada uma dessas etapas serão detalhados nas próximas seções.

Algorithm 1 TemplateGenerator

1: **Entrada:**
   Lista de instâncias X, classificador C, conjunto de classificadores $\mathbb{O}$
2: **Saída:**
   Lista de templates $\mathbb{T}$, conjunto de lexicons $\mathbb{L}$
3: **Inicialização:**
   $\mathbb{T} \leftarrow \varnothing$, ▷ conjunto de templates gerados
   $\mathbb{L} \leftarrow \varnothing$, ▷ conjunto de lexicons gerados
   $\mathbb{S} \leftarrow \varnothing$ ▷ conjunto de sentenças geradas
4: **for** $x_k \in X$ **do**
5: $\mathbb{S} \leftarrow \mathbb{S} \cup S_k$, $S_k =$ sentenças geradas a partir de x_k
6: **end for**
7: **for** $s_i \in \mathbb{S}$ **do**
8: **for** $t_j \in tokens(s_i)$ **do**
9: $R(s_i, t_j) \leftarrow$ ranking do token t_j na predição de s_i
10: **end for**
11: **end for**
12: $\mathbb{S} \leftarrow FILTRO(\mathbb{S})$
13: **for** $s_i \in \mathbb{S}$ **do**
14: $\mathbb{W} = n$ tokens mais bem rankeados
15: $s_i* = [t_0*, \ldots, t_n*], t_j* = \{lexicon\_name\}$ se $t_j \in \mathbb{W}$, senão t_j
16: $\mathbb{T} \leftarrow \mathbb{T} \cup s_i*$
17: $\mathbb{L} \leftarrow \mathbb{L} \cup \mathbb{W}$
18: **end for**
   return $\mathbb{T}, \mathbb{L}$

3.1.1 Quebrando instâncias em sentenças

A etapa inicial do algoritmo é quebrar as instâncias em sentenças, como mostram as linhas 4-6 do algoritmo. Isto significa que, cada instância x_k será decomposta em um conjunto de sentenças menores S_k e, cada uma dessas sentenças, fará parte de um único conjunto $\mathbb{S}$ composto por todas as sentenças geradas. Na Figura 6 podemos ver uma instância original sendo dividida em três sentenças, representadas na figura pelas diferentes cores dos retângulos.

Figura 6 – Exemplo de instância sendo dividida em sentenças

Text	Label
This is a really good flick with awesome humor. I bought this movie right after i saw it. Great directing, good script, worth renting.	1 (pos)

Fonte: Autoria própria.

A principal motivação para realizar essa transformação nos dados é que queremos utilizar os templates para testar capacidades linguísticas do modelo usando uma ferramenta de testes, e precisamos de instâncias com foco nesta capacidade específica. Instâncias maiores tendem a endereçar múltiplas capacidades linguísticas.

Para quebrar as instâncias em sentenças utilizamos o **NLTK** (NLTK. . . , 2022), uma biblioteca desenvolvida em Python especializada em tarefas de processamento de linguagem natural.

Ainda dentro desta etapa, cada sentença é classificada pelo *Oráculo*. O *Oráculo* é formado por um conjunto de modelos auxiliares especializados na mesma tarefa do modelo alvo. A classificação feita pelo *Oráculo* é feita usando os resultados obtidos por cada modelo que o compõe, onde a classe é definida como sendo a mais frequente ou, em caso de empate, a que resulta em uma maior média no *score* das predições.

Com as sentenças geradas, cada sentença s_i é dividida em tokens, onde para cada token t_j é calculado o ranking $R(s_i, t_j)$ para descobrir quais as palavras são mais relevantes na predição. Este passo está representado no algoritmo pelas linhas 7-11. Cada *token* gerado neste passo também é classificado pelo *Oráculo*.

3.1.2 Filtrando as sentenças

Normalmente, a etapa de transformação das instâncias em sentenças resulta em um conjunto de sentenças que podem ou não estarem relacionadas com a capacidade linguística que queremos testar. Isto significa que apenas um subconjunto dessas sentenças deve se tornar um conjunto de *templates* aplicável. A Figura 7 ilustra um cenário onde uma das sentenças de uma instância (a destacada em verde) passa pelos filtros. Nesta etapa, apenas os "bons candidatos" a se tornarem templates devem permanecer.

Figura 7 – Exemplo de sentença selecionada para ser transformada em template

Text	Label
This is a really good flick with awesome humor. I bought this movie right after i saw it. Great directing, good script, worth renting.	1 (pos)

Fonte: Autoria própria.

Podemos considerar um "bom candidato" a *template* qualquer sentença que tende a endereçar a capacidade linguística que queremos testar. Por exemplo, na figura está selecionada uma sentença que expressa de forma clara uma opinião positiva, que pode ser útil para testarmos a capacidade vocabulário. Em um outro cenário, se quisermos testar negação de opiniões *positivas* com um modelo de análise de sentimentos, qualquer sentença *positiva* pode ser considerada um "bom candidato".

A aplicação de filtros (linha 12 do algoritmo) busca então eliminar as sentenças de menor potencial para testar a capacidade linguística do modelo. Note que a combinação dos filtros usados está diretamente relacionada com a capacidade linguística que queremos testar.

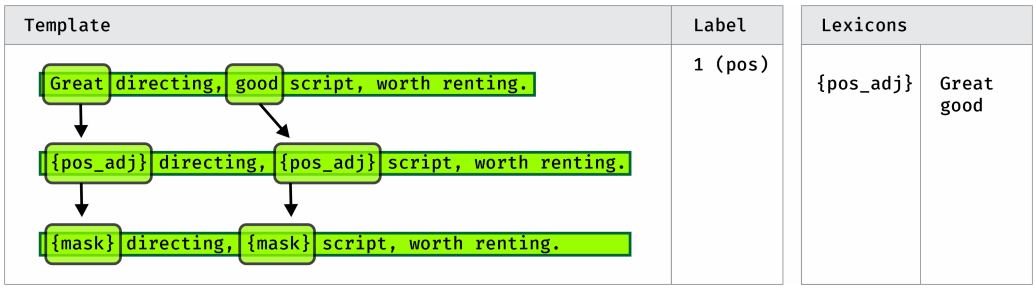
3.1.3 Transformando as sentenças em templates

A transformação das sentenças em templates é feita através da substituição de algumas palavras por máscaras, que chamaremos de *lexicons*. Esta etapa está representada no algoritmo pelas linhas 13-18. A escolha dessas palavras é feita de modo a selecionar as n palavras mais relevantes na predição da sentença, representadas no algoritmo pelo conjunto $\mathbb{W}$, e substituí-las por identificadores de *lexicons*. Após a substituição obtemos s_i*, que terá as máscaras no lugar das palavras mais relevantes. Para descobrir quais as palavras são mais relevantes é utilizado o ranqueamento das palavras realizado em etapas anteriores.

Uma vez substituídas, as palavras relevantes são armazenadas em conjuntos de *lexicons*, que podem ser utilizados durante a execução com a ferramenta de testes para substituição nos templates gerados. O **TemplateGenerator** produz três versões da mesma sentença: uma delas é a versão original da sentença, uma outra com o *lexicons* já definidos (como {pos_adj} por exemplo), e uma terceira com *lexicons* genéricos do tipo {mask}.

A figura 8 ilustra um template produzido pelo **TemplateGenerator**. Note que, além de produzir os templates, o algoritmo também produz um conjunto de *lexicons*. A primeira versão mostrada nesta figura é a própria sentença original, que poderá servir como comparativo com os casos testes gerados.

Figura 8 – Exemplo de template extraído a partir de uma instância



Fonte: Autoria própria.

Uma outra versão é gerada com os *lexicons* substituindo as palavras mais relevantes. Esta versão da sentença é a que será usada para geração de casos de teste pela ferramenta de teste. A ferramenta de testes substituirá cada *lexicon* por palavras que pertencem à lista de candidatas para cada *lexicon*.

Apesar do **TemplateGenerator** gerar também um conjunto de *lexicons*, é possível usar um conjunto criado separadamente. Para isso, podemos fazer uso da terceira e última versão da sentença gerada, a qual podemos usá-la como entrada para um modelo de linguagem mascarada (**BERT** por exemplo) para gerar palavras que se "encaixam" no lugar de {mask}, servindo como base para construção de um conjunto de *lexicons*.

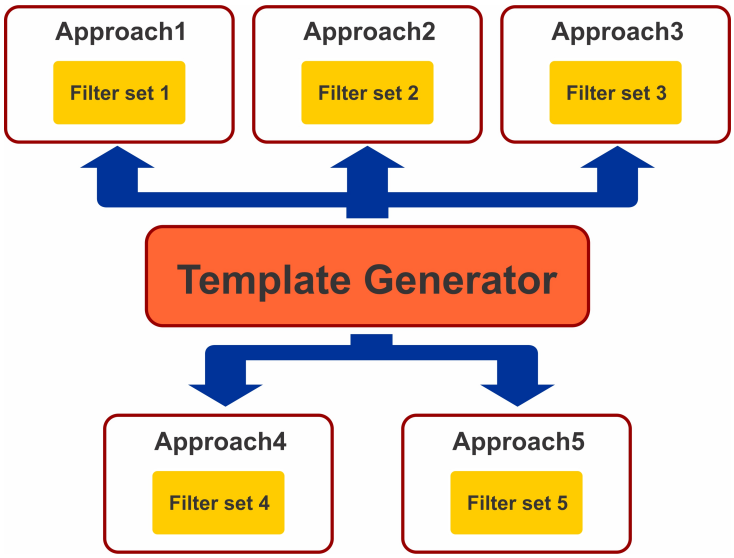
O conjunto de *lexicons* gerado é formado por todas as palavras retiradas de todos os templates gerados, categorizadas por tipo de *lexicon*. No exemplo da figura, como temos apenas um *template* gerado, temos apenas as palavras *Great* e *good* transformadas em *lexicons*.

3.2 Variações da Abordagem

Como vimos no **Algoritmo 1**, existe uma etapa chamada **FILTRO** que nos permite aplicar diversas combinações de diferentes filtros. Com esta possibilidade, foram desenvolvidas cinco diferentes abordagens, as quais divergem entre si principalmente nos tipos de filtros utilizados.

A Figura 9 mostra uma abstração das cinco abordagens derivadas do **Template Generator**. Note que cada variação possui um conjunto de filtros específico, caracterizando cada uma das abordagens.

Figura 9 – Variações das abordagens



Fonte: Autoria própria.

As etapas explicadas nas seções anteriores são comuns a todas as abordagens desenvolvidas. Uma característica importante é que elas foram desenvolvidas de modo a suportar alguns parâmetros de configuração. Devido a variabilidade nos filtros usados em cada abordagem, os parâmetros também podem variar entre as abordagens.

Por exemplo, as abordagens 3, 4 e 5 utilizam um parâmetro $score_{min}$, que limita inferiormente o *score* de classificação obtido pelas predições feitas internamente pelo **Oráculo**. Já a abordagem 5 necessita de um parâmetro L_{min}, que limita o número mínimo de palavras de um template.

Dois importantes parâmetros são comuns a todas as abordagens: n e m. O parâmetro n indica o número de *lexicons* que o template terá. Já o parâmetro m indica o número de palavras relevantes que o algoritmo usará como candidatos a *lexicons*. Mais detalhes sobre cada uma das variações da abordagens são explanadas nas seções seguintes.

3.2.1 Abordagem 1

A **Abordagem1** tem como primeiro passo o ranqueamento das palavras da instância completa. Em seguida essa instância é quebrada em sentenças.

Ao iniciar a etapa de filtros, o primeiro filtro aplicado é uma seleção das sentenças que contêm ao menos uma das p palavras mais bem ranqueadas na etapa anterior. Para este filtro, o valor de p foi definido como sendo o número de sentenças geradas ao quebrar a instância, o que significa que, na melhor das hipóteses, cada sentença conterá uma dessas palavras, fazendo com que todas elas passem nesse filtro.

Após isso, são selecionadas apenas as sentenças em que, dadas as n palavras mais relevantes da sentença, pelo menos m palavras pertencem a um dado conjunto de classes gramaticais. As sentenças que sobrarem são submetidas a uma etapa de classificação pelo **Oráculo**. Por fim, um novo filtro é aplicado, onde são selecionadas as sentenças classificadas de forma unânime pelo oráculo. A classificação é considerada unânime quando todos os modelos do **Oráculo** classificam a instância com a mesma classe.

3.2.2 Abordagem 2

Na **Abordagem2** são aplicados passos semelhantes aos da **Abordagem1**. A principal diferença é que acrescentamos uma etapa antes de filtrar as sentenças com as palavras mais relevantes pertencentes a um conjunto de classes gramaticais.

Essa nova etapa consiste na aplicação de um novo ranqueamento das palavras, agora considerando apenas as palavras de cada sentença. O principal impacto desta mudança é no comportamento do filtro por classes gramaticais. Com este ranqueamento, o algoritmo passa a considerar (a partir deste momento) a importância de cada palavra na predição da **sentença**, ao invés da **instância completa** como acontece na **Abordagem1**.

3.2.3 Abordagem 3

Diferentemente das abordagens anteriores, a **Abordagem3** já inicia com a divisão da instância em sentenças. O **Oráculo** entra então em cena, classificando cada uma das sentenças geradas na etapa anterior.

O primeiro filtro aplicado preserva apenas as sentenças classificadas de forma unânime pelo oráculo. Para refinar ainda mais o conjunto de sentenças em relação aos resultados do **Oráculo**, um novo filtro seleciona apenas as sentenças que atingem um *score* médio maior que um $score_{min}$ considerando as classificações feitas por todos os modelos do **Oráculo**.

Em seguida, apenas as sentenças que possuem suas L palavras mais relevantes pertencentes a um dado conjunto de classes gramaticais (a exemplo das abordagens anteriores) são selecionadas. Finalmente, aplicamos mais um refinamento, onde um novo filtro escolhe apenas as sentenças em que as L relevantes mencionadas no filtro anterior possuem (quando classificadas individualmente) um *score* maior que um $score_{min}$.

Este filtro de refinamento pode ser útil para evitar sentenças/palavras classificadas próximo da fronteira de decisão, o que pode ajudar na geração de templates com foco em algumas capacidades linguísticas específicas. Por exemplo, se quisermos testar o vocabulário trocando palavras por sinônimos, queremos trocar um adjetivo positivo por outro adjetivo positivo. Sem este filtro corremos o risco de um adjetivo negativo ser classificado (erroneamente) como positivo, mas próximo da fronteira de decisão. Isto resultará em um caso de teste errado.

3.2.4 Abordagem 4

Assim como ocorre entre as duas primeiras abordagens, a **Abordagem4** é bem semelhante à **Abordagem3**, existindo apenas algumas diferenças sutis entre elas.

Antes das etapas já descritas para a **Abordagem3**, um filtro inicial é aplicado nas instâncias completas, selecionando apenas aquelas que são classificadas unanimemente pelo **Oráculo**. Daí em diante segue o mesmo fluxo da **Abordagem3**.

3.2.5 Abordagem 5

A **Abordagem5**, inicia com a divisão das instâncias em sentenças. Em seguida as palavras são ranqueadas de acordo com sua importância dentro de cada sentenças gerada.

Um primeiro filtro elimina todas as sentenças com tamanho menor do que L_{min} palavras. Este filtro pode ser útil para eliminar sentenças muito pequenas. Na sequência, o filtro que seleciona as sentenças que possuem as n palavras mais relevantes pertencentes a um dado conjunto de classes gramaticais é aplicado.

No final, apenas as sentenças em que o *score* obtido na predição pelo oráculo ultrapassa o valor de $score_{min}$ são escolhidas, a exemplo das abordagens 3 e 4.

4 ESTUDO DE CASO

Neste capítulo iremos tratar como foi realizado o estudo de caso com o **IMDB**. Iniciaremos definindo quais são nossas questões de pesquisa. Em seguida, na seção metodologia, veremos como foi realizado nosso experimento, onde explicaremos detalhes como os modelos utilizados, *dataset*, abordagens usadas e configuração do ambiente de execução. Após isso, veremos os resultados obtidos. Por fim, faremos uma breve discussão acerca desses resultados.

4.1 Questões de Pesquisa

Como mencionado antes, nosso principal objetivo é automatizar uma parte da atividade de testes com a ferramenta **CheckList** que ainda é bastante manual: a geração de templates para geração de dados de teste. Diante disso, seguem as questões de pesquisa que norteiam nosso estudo.

4.1.1 Questão de Pesquisa 1: É possível extrair, a partir de dados reais provenientes de datasets, templates com foco em capacidades linguísticas?

Esta é a nossa principal questão de pesquisa, pois a viabilidade do restante do trabalho depende de uma resposta positiva para essa pergunta.

Para responder esta pergunta, realizaremos um estudo de caso onde adaptações das abordagens serão desenvolvidas para ter como foco uma determinada capacidade linguística (ver seção 3.1 para ver mais sobre capacidades linguísticas). Aplicaremos então as abordagens adaptadas a um modelo de análise de sentimentos e um *dataset* de testes.

4.1.2 Questão de Pesquisa 2: É possível automatizar a tarefa de geração de templates?

O estudo de caso também servirá como base para respondermos esta questão, uma vez que as abordagens (e consequentemente as adaptações mencionadas anteriormente) são desenvolvidas com a proposta de extrair os *templates* do *dataset* de forma automatizada.

4.1.3 Questão de Pesquisa 3: Quão eficiente conseguimos ser com nossas abordagens?

Para respondermos a esta questão, executaremos as cinco variações da abordagem, observando o tempo de execução de cada uma delas. Neste momento entrará em cena uma abordagem **Random**, que será desenvolvida com o único propósito de servir como base de comparação para as abordagens desenvolvidas.

4.1.4 Questão de Pesquisa 4: Quão eficaz conseguimos ser com nossas abordagens?

Para responder a esta pergunta usaremos outras métricas coletadas durante as execuções das abordagens, como, por exemplo, a quantidade de falhas que a a ferramenta de teste conseguiu

encontrar no modelo usando como base os templates gerados por cada abordagem. Mais uma vez usaremos aqui a abordagem **Random** como comparativo de qualidade em relação às abordagens desenvolvidas.

4.2 Metodologia

Esta seção descreve a escolha dos modelos e *dataset* do estudo, bem como a seleção das instâncias, as abordagens com suas particularidades, a ferramenta de teste e o ambiente de execução.

4.2.1 Modelos de Análise de Sentimentos

Para o estudo de caso foram usados alguns modelos de análise de sentimentos, sendo um deles como modelo alvo e outros quatro para compor o *Oráculo* do **TemplateGenerator**. Estes modelos fazem parte do conjunto de modelos de benchmark do framework **TextAttack** (MORRIS et al., 2020), disponíveis no Hugging Face. Todos os modelos usados foram pré-treinados e, posteriormente, ajustados usando o **TextAttack** para tarefa de classificação com o dataset **IMDB**.

O principal modelo usado, o nosso modelo alvo, é o **bert-base-uncased-imdb**. Este modelo foi treinado por 5 épocas com *batch-size* igual a 16, uma taxa de aprendizagem de 2×10^{-5} e um comprimento máximo de sequência de 128. Uma característica deste modelo é que ele não faz distinção entre letras maiúsculas e minúsculas.

O modelo **albert-base-v2-imdb**, um dos modelos que compõem o *Oráculo*, foi treinado por 5 épocas com *batch-size* igual a 32, uma taxa de aprendizagem de 2×10^{-5} e um comprimento máximo de sequência de 128.

O **distilbert-base-uncased-imdb**, outro modelo que compõe o *Oráculo*, foi treinado por 5 épocas com *batch-size* igual a 16. A taxa de aprendizagem usada foi de 2×10^{-5} e o comprimento máximo de sequência de 128. Assim como **bert-base-uncased-imdb**, este modelo não faz distinção entre letras maiúsculas e minúsculas.

O modelo **roberta-base-imdb**, mais um modelo que compõe o *Oráculo*, foi treinado por 5 épocas com *batch-size* igual a 64. A taxa de aprendizagem usada foi de 3×10^{-5} e o comprimento máximo de sequência de 128.

O modelo **xlnet-base-cased-imdb**, nosso último modelo, é o único dos modelos selecionados que não é baseado em **BERT**. Ele foi treinado por 5 épocas com *batch-size* igual a 32. A taxa de aprendizagem usada foi de 2×10^{-5} e o comprimento máximo de sequência de 512.

A Tabela 1 mostra um resumo das características já citadas para modelos usados, além das acurácias de cada um deles.

Tabela 1 – Resumo das características dos modelos usados no estudo de caso.

Modelo	Épocas	Batch-size	Tx aprend.	Compr. seq.	Acurácia
bert-base-uncased-imdb	5	16	2×10^{-5}	128	91,9%
albert-base-v2-imdb	5	32	2×10^{-5}	128	91,3%
distilbert-base-uncased-imdb	5	16	2×10^{-5}	128	90,3%
roberta-base-imdb	5	64	3×10^{-5}	128	94,1%
xlnet-base-cased-imdb	5	32	2×10^{-5}	512	95,7%

4.2.2 Dataset

O dataset usado foi o conjunto de dados do **IMDB**. O **IMDB** é um *benchmark* amplamente usado em tarefas de análise binária de sentimento, contendo 50000 amostras de resenhas de filmes, escritas em inglês e distribuídas igualmente entre duas classes distintas. O **IMDB** é composto por dois subconjuntos: um subconjunto de dados de treino, com 25000 amostras e um subconjunto de dados de teste, com as outras 25000 amostras (MAAS et al., 2011).

Cada instância deste dataset é rotulada como classe 0 (negativa) ou classe 1 (positiva). A Figura 10 mostra um exemplo de instância positiva do subconjunto de teste do **IMDB**.

Figura 10 – Exemplo de instância do dataset IMDB

Text	Label
This is a really good flick with awesome humor. Jim Verney as we know was very good with facial expressions and demonstrates a lot of it in this movie.This is definitely the best of the Ernest films.I would surely recommend it to any Ernest fan out there.i find myself to have great taste in movies and I'm sure anyone will enjoy this movie. In the movie ,(Ernest) plays 2 roles, bad guy and good guy and plays them quite well. I really enjoy exaggeration type humor where things just seem impossible,like in the naked gun films for example, and there is plenty of it in this movie.I bought this movie right after i saw it. Good directing, good script, worth renting.	1 (pos)

Fonte: Autoria própria.

Para a execução descrita neste trabalho foram usadas apenas o subconjunto de dados de teste do **IMDB**, que contém 25000 amostras, conforme citado anteriormente.

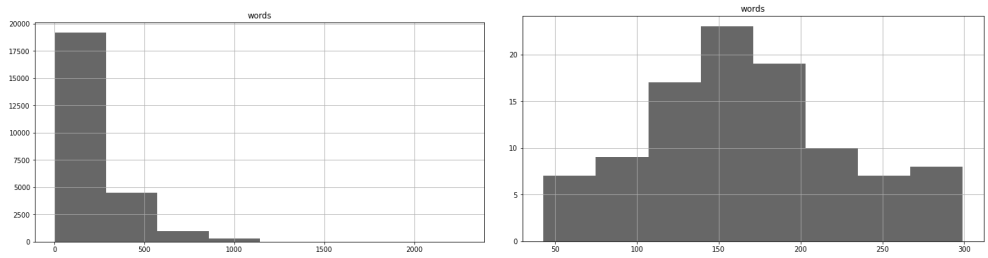
4.2.2.1 Seleção de Instâncias

Embora tenhamos usado neste trabalho apenas o subconjunto de dados de teste do **IMDB**, este ainda assim possui uma quantidade relativamente grande de instâncias (25.000 instâncias, divididas igualmente entre positivas e negativas), além de possuir também uma grande variabilidade na quantidade de palavras. Por exemplo, a menor instância possui 4 palavras, ao passo que a maior possui 2278 palavras.

Diante disso foi necessário reduzir a amostra para uma quantidade que conseguíssemos trabalhar, no caso, 100 instâncias, todas com tamanho entre 10 e 300 palavras. Essa redução foi feita reduzindo-se o dataset original para conter apenas as instâncias com o tamanho dentro da faixa mencionada anteriormente.

Após a primeira redução, escolhemos as 100 amostras de forma aleatória, mantendo também uma divisão equilibrada entre as duas classes. Com isso, conseguimos uma melhor distribuição (próxima da normal) para a quantidade de palavras, como mostra a Figura 11.

Figura 11 – Distribuição por número de palavras para o dataset original **(a)** e para o conjunto de 100 amostras **(b)**



(a) Dataset original (b) Dataset amostrado

Fonte: Autoria própria.

4.2.3 Usando as abordagens com o IMDB

O experimento realizado neste estudo de caso consiste em extrair *templates* a partir de um *dataset* de testes do **IMDB**, com foco na capacidade linguística **Vocabulário**.

A estratégia usada foi gerar templates que trocam algumas palavras de uma frase por outras de modo a manter o contexto. As palavras a serem trocadas são escolhidas levando em consideração sua classe gramatical dentro do texto. Como o modelo alvo é treinado para realizar a tarefa de análise de sentimentos, escolhemos como classes gramaticais relevantes os **verbos** e **adjetivos**, já que estes podem ser positivos ou negativos.

Para gerar os *templates*, executamos as cinco diferentes variações do algoritmo original (ver seção 3.2). Alguns parâmetros de configuração foram usados para a execução de cada uma delas, dentre eles estão o número de palavras relevantes W_R usadas e o número de palavras substituídas por *lexicons L*. Basicamente, o algoritmo escolhe as L substituições a partir W_R palavras candidatas, desde que atendam os critérios de classe gramatical definidos (verbos ou adjetivos). Para todas as execuções usamos os valores $W_R = 4$ e $L = 2$.

Um componente importante do algoritmo é o que chamamos de **Oráculo**, que é utilizado, por exemplo, para decidir qual será classe das instâncias de teste que serão produzidas a partir do *template* gerado. Para este estudo de caso, utilizamos como oráculo (em todas as abordagens) um conjunto formado pelos seguintes modelos: **albert-base-v2-imdb**, **distilbert-base-uncased-imdb**, **roberta-base-imdb** e **xlnet-base-cased-imdb**. Para mais detalhes sobre esses modelos, consultar a seção 4.2.1.

Outra característica comum em todas as abordagens executadas é a estratégia de ranqueamento de palavras. Neste experimento foi utilizada a estratégia **Replace-1 Score** (GAO et al., 2018a) em todas as execuções.

Alguns outros parâmetros de configuração utilizados são particulares para algumas abordagens. Nas Abordagens 3,4 e 5 utilizamos um parâmetro $score_{min} = 0,8$, valor este que foi obtido empiricamente. Para a abordagem 5 utilizamos, além dos já mencionados, o parâmetro $W_{min} = 5$. O significado parâmetros estão explicados na seção 3.2.

Além das abordagens já discutidas anteriormente, uma abordagem adicional foi desenvolvida, onde alguns critérios usados nos filtros foram aleatorizados. Nos referiremos a esta abordagem como abordagem **Random**. Esta abordagem foi necessária para estabelecermos um parâmetro de comparação para as demais abordagens.

4.2.4 Ferramenta de Teste

A ferramenta de testes para **PLN** que está no cerne deste trabalho é o **CheckList**. Desta maneira, os templates gerados de forma automatizada por nossas abordagens serão usados como entrada para o **CheckList**, que se encarregará de executar os casos de teste baseados nos nossos *templates*.

A maneira como o **CheckList** utiliza os *templates* basicamente é como "receitas"para geração de dados de teste fictícios, utilizados então como casos de teste dentro de a suíte de testes.

4.2.5 Ambiente de Execução

Como ambiente de execução, foi utilizada uma máquina com sistema operacional Windows 10, com 16GB de memória RAM e CPU Intel Core I7. Todas as abordagens foram executadas com o auxílio da ferramenta Jupyter Notebook, uma plataforma de desenvolvimento interativo para linguagem Python baseado em ambiente web.

4.3 Resultados

Após a execução de cada uma das abordagens obtivemos os resultados mostrados na Tabela 2. As execuções para cada abordagem foram realizadas fornecendo como entrada uma mesma amostra de 100 instâncias do *dataset* de testes **IMDB**.

Tabela 2 – Templates gerados por abordagem

Abordagems	Tempo de Execução	Templates Gerados	Tempo médio/Template
Abordagem1	123m 6,7s	33	3min 43,8s
Abordagem2	132m 12,5s	35	3min 46,6s
Abordagem3	21m 55,9s	18	1min 13,1s
Abordagem4	22m 43,2s	17	1min 20,2s
Abordagem5	28m 41,8s	26	1min 6,2s
Random	1m 10,3s	18	0min 3,9s

A **Questão de Pesquisa 1** pode ser respondida com base nos resultados obtidos na Tabela 2. Os resultados mostram que é possível sim extrair *templates* a partir de dados reais provenientes de *datasets*. Pelos dados mostrados podemos ver que a **Abordagem2** foi a que produziu mais templates (35), enquanto a **Abordagem4** foi a que menos produziu (17). Para este quesito, a abordagem **Random** não serve de comparativo pois, devido aos critérios de filtragem serem aleatórios, o número de *templates* a serem gerados é um parâmetro configurável nesta abordagem.

Todo o procedimento para geração dos *templates* é completamente automatizado, onde temos apenas que fornecer os dados e o modelo para o **TemplateGenerator**. Isso responde nossa **Questão de Pesquisa 2**. Não só conseguimos gerar os templates de forma automatizada, mas também o conjunto de *lexicons*.

Em nossa **Questão de Pesquisa 3** somos questionados quanto a eficiência das abordagens. Ainda de acordo com a Tabela 2, podemos ver que o tempo de execução (**TE**) varia bastante de acordo com a abordagem, sendo que a abordagem **Random** obteve o melhor tempo. Isto já era esperado, visto que os critérios de filtragem para geração de *templates* são randômicos.

Dentre as abordagens em análise, podemos ver que a **Abordagem 3** foi a mais eficiente, isso se considerarmos os números de forma absoluta. No entanto, quando olhamos para o tempo de execução e o número de *templates* gerados, juntos, percebemos que a **Abordagem 5** foi a mais eficiente, gerando 26 templates em 28min 41.8s.

Para responder nossa última (e mais complexa) questão de pesquisa, a **Questão de Pesquisa 4**, faremos uma análise sobre a qualidade dos *templates* gerados. Inicialmente vamos olhar para a Tabela 3. Nela podemos ver o total de casos de testes gerados pela ferramenta de testes com os *templates* produzidos por cada abordagem, além de quantos desses casos de teste passaram e quantos falharam.

Uma primeira análise que podemos fazer é em relação ao percentual de falhas encontradas usando cada abordagem. Olhando apenas as informações da Tabela 3, poderíamos concluir que a **Abordagem 2** foi a mais eficaz, sendo aquela que levou a ferramenta de testes a encontrar mais falhas no modelo, tanto em números absolutos quanto em percentuais. No entanto, embora o **TemplateGenerator** tenha conseguido seu objetivo, alguns desses resultados podem não representar a realidade, podendo ser, por exemplo, *Falsos Positivos*.

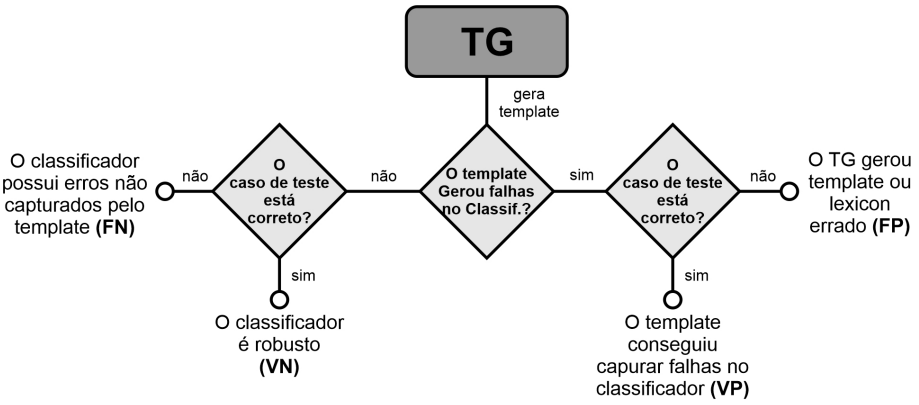
Tabela 3 – Casos de teste gerados por abordagem

Abordagems	Casos de Teste	Passando	Falhando
Abordagem1	5839	3819 *(65,4%)*	2020 *(34,6%)*
Abordagem2	6197	3745 *(60,4%)*	2452 *(39,6%)*
Abordagem3	1002	847 *(84,5%)*	155 *(15,5%)*
Abordagem4	892	749 *(84,0%)*	143 *(16,0%)*
Abordagem5	2232	1815 *(81,3%)*	417 *(18,7%)*
Random	508	342 *(67,3%)*	166 *(32,7%)*

Dito isso, precisamos então definir o que são *Falsos Positivos* (**FP**), *Verdadeiros Positivos* (**VP**), *Falsos Negativos* (**FN**) e *Verdadeiros Negativos* (**VN**) dentro do nosso contexto.

Consideramos três possíveis situações para definir um caso como *Falso Positivo* ou *Falso Negativo*: a) a instância não expressa opinião; b) a instância foi gerada com a classe errada (*template errado*); c) a instância foi gerada com substituições de palavras erradas (*lexicons errados*). Os critérios de decisão para cada um dos tipos de resultado mencionados anteriormente estão resumidos no diagrama da Figura 12.

Figura 12 – Fluxo de decisão para Falsos Negativos, Verdadeiros Negativos, Falsos Positivos e Verdadeiros Positivos



Fonte: Autoria própria.

Note que, de acordo com o diagrama, os casos de sucesso do **TemplateGenerator** são definidos basicamente pelos *Verdadeiros Positivos*. Note também, que o conjunto formado pelos *Verdadeiros Positivos* e *Verdadeiros Negativos* é o conjunto de casos de teste corretos, ou seja, gerados por templates **corretos**.

Para viabilizar a avaliação selecionamos amostras dos casos de teste utilizando a técnica de *Stratified Random Sampling* (Amostragem Aleatória Estratificada), gerando um subconjunto de casos de teste representativo com uma taxa de 5% de erro amostral e nível de confiança de 95%.

Todos os resultados amostrados foram submetidos a uma análise onde destacamos os resultados verdadeiramente relevantes. Cada um deles foi classificado em um dos quatro possíveis tipos de resultado: falso negativo (**FN**), verdadeiro negativo (**VN**), falso positivo (**FP**) ou verdadeiro negativo (**VN**), que pode ser visto na Tabela 4.

A eficácia mostrada na Tabela 4 é dada pela quantidade de falhas verdadeiras (**VP**) em relação ao número de casos de teste da amostra. Após essa análise mais detalhada dos resultados, podemos finalmente responder nossa **Questão de Pesquisa 4**. Segundo os resultados na tabela, a **Abordagem 3** foi a mais eficaz (com 26,5%), seguida de perto pelas **Abordagem 4** (24,2%) e **Abordagem 5** (23,8%).

Tabela 4 – Avaliação dos resultados com 100 instâncias

Abordagems	Amostras	Passando	Falhando	FN	VN	FP	VP	Eficácia
Abordagem1	479	246	233	98	148	178	55	*11,5%*
Abordagem2	488	248	240	73	175	151	89	*18,2%*
Abordagem3	299	199	100	36	163	22	78	*26,1%*
Abordagem4	285	191	94	19	172	25	69	*24,2%*
Abordagem5	390	232	158	25	207	65	93	*23,8%*
Random	257	152	105	75	77	103	2	*0,8%*

4.4 Discussão dos resultados

Como podemos ver anteriormente na Tabela 4, todas as abordagens superaram a abordagem **Random** no quesito *Eficácia* (*Verdadeiros Positivos* em relação ao número de amostras). Podemos ver também, que as abordagens 3, 4 e 5 foram superiores às abordagens 1 e 2, tanto no quesito *Eficácia* quanto no *Tempo de Execução* (ver Tabela 2).

Uma outra análise importante a ser considerada é em relação aos *Falsos Positivos* e *Falsos Negativos*. De acordo as definições mencionadas anteriormente, existem três condições que podem levam a classificação de um caso como *Falso Positivo* ou *Falso Negativo*. Podemos ver na Tabela 5 um resumo de quais foram os motivos considerados para esses casos.

Tabela 5 – Motivos por abordagem para falsos positivos e falsos negativos

Abordagems	Falsos negativos				Falsos positivos			
	Total	NO	TE	LE	Total	NO	TE	LE
Abordagem1	98	18	34	46	178	34	74	70
Abordagem2	73	13	11	49	151	40	33	78
Abordagem3	36	16	0	20	22	0	0	22
Abordagem4	19	0	0	19	25	0	0	25
Abordagem5	25	15	4	6	65	0	32	33
Random	75	39	27	9	103	42	53	8

Para os Falsos Negativos, podemos dizer que o motivo *Template Errado* (**TE**) é mais crítico que os motivos *Lexicon Errado* (**LE**) e *Não é Opinião* (**NO**), já que esse pode levar a não-detecção de problemas no classificador. Já para os Falsos Positivos, os motivos **NO** e **TE** são críticos, pois indicam problemas no template.

Analisando os dados da tabela para Falsos Negativos, podemos notar que a **Abordagem1** obteve resultado para **TE** proporcionalmente semelhante à abordagem **Random**, ao passo que as abordagens **Abordagem3** e **Abordagem4** obtiveram os melhores resultados para este tipo de motivo. Ainda de acordo com a tabela, as abordagens **Abordagem3** e **Abordagem4** também obtiveram os melhores resultados considerando agora os Falsos Positivos, onde 100% dos casos tiveram como motivo **LE**, que é menos crítico que os demais.

4.5 Ameaças à Validade

Durante o experimento realizado com o **IMDB** pudemos observar alguns aspectos que poderiam levar-nos a resultados diferentes.

Em primeiro lugar, o próprio ambiente de execução já pode ser considerado uma dessas variáveis. Uma máquina com maior poder de processamento poderia levar a tempos de execução menores, embora devesse manter uma tendência proporcional em relação aos tempos obtidos.

Nós mencionamos que o modelo utilizado como modelo alvo no experimento foi o **bert-base-uncased-imdb**, além de outros quatro modelos que foram usados como **Oráculo** para o **TemplateGenerator**. Se trocássemos o modelo alvo por um dos usados no **Oráculo** poderíamos influenciar diretamente nos resultados. Um modelo diferente poderia, por exemplo, fazer com que uma outra abordagem fosse a mais eficaz. A estratégia utilizada depende também da qualidade dos modelos usados no Oráculo. A ausência de modelos bem treinados pode atrapalhar os resultados das abordagens.

O *dataset* usado também é um fator que poderia influenciar nos resultados. Se usássemos um *dataset* com instâncias menores, em que as instâncias são simples sentenças, poderíamos favorecer (ou desfavorecer) abordagens que executam algum filtro antes que dividir as instâncias em sentenças. Isso porque, como a instância é uma mera sentença, a divisão em sentenças resultará na própria instância, não causando efeito algum.

Outra questão importante é que, no experimento realizado neste trabalho, nós focamos em uma capacidade linguística específica (**vocabulário**). É possível que para uma capacidade linguística diferente tenhamos desempenhos diferentes.

4.6 Considerações finais

Durante este capítulo vimos um estudo de caso do **TemplateGenerator**, onde definimos e respondemos nosssas questões de pesquisa realizando uma aplicação experimental com o *dataset* **IMDB**. Juntamente com o *dataset*, utilizamos alguns modelos treinados para tarefa de análise de sentimento, sendo o **bert-base-uncased-imdb** o modelo alvo e outros quatro modelos usados como **Oráculo** auxiliar do **TemplateGenerator**.

Os resultados foram obtidos e usados como base para responder às questões de pesquisa. Comparamos então os resultados das abordagens com os obtidos com uma abordagem **Random** e avaliamos as qualidades dos *templates* gerados. Introduzimos também, os conceitos de *Falsos Negativos*, *Verdadeiros Negativos*, *Falsos Positivos* e *Verdadeiros Positivos*, que nos ajudaram a explicar alguns resultados.

5 CONCLUSÃO E TRABALHOS FUTUROS

A utilização de algumas ferramentas para criação e execução de suítes de testes para modelos de **PLN** normalmente inclui tarefas manuais que podem ser custosas, como a elaboração de *templates* para geração de casos de teste, por exemplo.

Nas seções anteriores discutimos a criação de uma abordagem automatizada para extrair *templates* e gerar casos de teste a partir de *datasets* como entrada, *templates* estes que focam em capacidades linguísticas de modelos de processamento de linguagem natural. A esta abordagem chamamos de **TemplateGenerator**.

A abordagem inicial deu origem a cinco diferentes variações, as quais foram submetidas a um estudo de caso, onde pudemos analisar os resultados a partir de um exemplo prático de uso.

De acordo com esses resultados, pudemos observar que as abordagens conseguiram uma eficácia que variou entre 11,5% e 26,1%, contra 0,8% obtido por nossa abordagem **Random**, usada como *baseline*. Os resultados sugerem então que é possível extrair templates a partir de dados de teste.

Pudemos perceber também, na avaliação dos resultados, que alguns ruídos ainda estão presentes nos *templates* gerados. Alguns deles causados por imprecisões na classificação de palavras por parte da biblioteca **NLTK**, outros por erros na classificação de *templates* e *lexicons* por parte do oráculo. Este último apresenta-se como um grande desafio, pois dependemos de dispor de modelos bem treinados e precisos para compor o oráculo.

Um dos possíveis trabalhos futuros poderia ser a execução das abordagens desenvolvidas para outros *datasets*, onde poderíamos comparar os resultados obtidos com os novos resultados. Além disso, poderia também ser feita uma classificação dos bugs encontrados nos modelos com os casos de testes gerados usando os templates. Outra possibilidade seria adaptar as abordagens para geração de *templates* com foco em capacidades linguísticas diferentes.

REFERÊNCIAS

AMERSHI, S. et al. Software engineering for machine learning: A case study. In: IEEE. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. [S.l.], 2019. p. 291–300. Citado na página 14.

BHATT, S. et al. A case study of efficacy and challenges in practical human-in-loop evaluation of NLP systems using checklist. In: *Proceedings of the Workshop on Human Evaluation of NLP Systems (HumEval)*. Online: Association for Computational Linguistics, 2021. p. 120–130. Disponível em: <https://aclanthology.org/2021.humeval-1.14>. Citado 5 vezes nas páginas 12, 17, 19, 24 e 25.

BRAGA, A. V. et al. Machine learning: O uso da inteligência artificial na medicina. *Brazilian Journal of Development*, v. 5, n. 9, p. 16407–16413, 2019. Citado 2 vezes nas páginas 11 e 14.

BROWN, T. B. et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. Citado na página 18.

DELUA, J. *Supervised vs. Unsupervised Learning: What's the Difference?* **IBM**, 2021. Accesso em: 15 nov. 2021. Disponível em: <https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning>. Citado 2 vezes nas páginas 14 e 15.

DEVLIN, J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: <https://aclanthology.org/N19-1423>. Citado 2 vezes nas páginas 11 e 15.

FENG, S. Y. et al. A survey of data augmentation approaches for NLP. *CoRR*, abs/2105.03075, 2021. Citado na página 20.

GAO, J. et al. Black-box generation of adversarial text sequences to evade deep learning classifiers. In: *2018 IEEE Security and Privacy Workshops (SPW)*. [S.l.: s.n.], 2018. p. 50–56. Citado 2 vezes nas páginas 20 e 36.

GAO, J. et al. Black-box generation of adversarial text sequences to evade deep learning classifiers. *CoRR*, abs/1801.04354, 2018. Citado na página 20.

GARG, S.; RAMAKRISHNAN, G. BAE: bert-based adversarial examples for text classification. *CoRR*, abs/2004.01970, 2020. Citado na página 20.

GOODFELLOW, I. J.; SHLENS, J.; SZEGEDY, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. Citado na página 20.

KIM, J.; FELDT, R.; YOO, S. *Guiding Deep Learning System Testing using Surprise Adequacy*. 2018. Citado na página 17.

KUMAR, A. *QA: Blackbox Testing for Machine Learning Models*. 2018. Accesso em: 16 nov. 2021. Disponível em: <https://dzone.com/articles/qa-blackbox-testing-for-machine-learning-models>. Citado na página 17.

LIU, Y. et al. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. Citado na página 15.

MAAS, A. L. et al. Learning word vectors for sentiment analysis. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, 2011. p. 142–150. Disponível em: <http://www.aclweb.org/anthology/P11-1015>. Citado na página 35.

MINAEE, S. *20 Popular Machine Learning Metrics. Part 1*: Classification regression evaluation metrics. 2019. Accesso em: 18 mai. 2022. Disponível em: <https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca.3e282a2ce>. Citado na página 11.

MORRIS, J. X. et al. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. *arXiv preprint arXiv:2005.05909*, 2020. Citado 3 vezes nas páginas 12, 21 e 34.

NLTK :: Natural Language Toolkit. 2022. Accesso em: 18 mai. 2022. Disponível em: <https://www.nltk.org/>. Citado na página 28.

OPENAI. *Attacking Machine Learning with Adversarial Examples*. 2017. Accesso em: 15 nov. 2021. Disponível em: <https://openai.com/blog/adversarial-example-research/>. Citado 2 vezes nas páginas 19 e 20.

OPENAI. *Improving Language Understanding with Unsupervised Learning*. 2018. Accesso em: 17 nov. 2021. Disponível em: <https://openai.com/blog/language-unsupervised/>. Citado na página 15.

PEI, K. et al. Deepxplore. *Proceedings of the 26th Symposium on Operating Systems Principles*, ACM, Oct 2017. Disponível em: <http://dx.doi.org/10.1145/3132747.3132785>. Citado na página 17.

RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. "why should i trust you?"explaining the predictions of any classifier. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. [S.l.: s.n.], 2016. p. 1135–1144. Citado 4 vezes nas páginas 12, 18, 19 e 21.

RIBEIRO, M. T. et al. Beyond accuracy: Behavioral testing of nlp models with checklist. *arXiv preprint arXiv:2005.04118*, 2020. Citado 4 vezes nas páginas 12, 22, 23 e 25.

RICCIO, V. et al. Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, Springer, v. 25, n. 6, p. 5193–5254, 2020. Citado na página 11.

ROUSSEAU, A.-L.; BAUDELAIRE, C.; RIERA, K. *Doctor GPT-3: hype or reality?* **Nabla**, 2020. Accesso em: 13 nov. 2021. Disponível em: <https://www.nabla.com/blog/gpt-3/>. Citado na página 18.

SANH, V. et al. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. Citado na página 15.

SAS. *Processamento de Linguagem Natural*: O que é e qual sua importância? 2022. Accesso em: 18 mai. 2022. Disponível em: <https://www.sas.com/pt_br/insights/analytics/processamento-de-linguagem-natural.html>. Citado na página 11.

SOMMERVILLE, I. *Engenharia de software*. Pearson Prentice Hall, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ>. Citado na página 11.

TAN, S. et al. *Reliability Testing for Natural Language Processing Systems*. 2021. Citado na página 17.

YANG, Z. et al. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019. Citado 2 vezes nas páginas 15 e 16.

YING, X. An overview of overfitting and its solutions. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2019. v. 1168, n. 2, p. 022022. Citado na página 12.

ZHANG, J. M. et al. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, IEEE, 2020. Citado 2 vezes nas páginas 11 e 16.

ZHOU, X. *Interpretability Methods in Machine Learning: A Brief Survey*. **Two Sigma**, 2021. Accesso em: 13 nov. 2021. Disponível em: <https://www.twosigma.com/articles/interpretability-methods-in-machine-learning-a-brief-survey/>. Citado na página 19.